R. Rios, J. A. Onieva, and J. Lopez, "HIDE_DHCP: Covert Communications Through Network Configuration Messages", *Proceedings of the 27th IFIP TC 11 International Information Security and Privacy Conference (SEC 2012)*, IFIP AICT vol. 376, pp. 162-173, 2012.
http://doi.org/http://dx.doi.org/10.1007/978-3-642-30436-1_14
NICS Lab. Publications: https://www.nics.uma.es/publications

HIDE_DHCP: Covert Communications Through Network Configuration Messages

Ruben Rios, Jose A. Onieva and Javier Lopez
Network, Information and Computer Security Lab
University of Malaga
29071, Spain
{ruben,onieva,jlm}@lcc.uma.es

Abstract

Covert channels are a form of hidden communication that may violate the integrity of systems. Since their birth in multilevel security systems in the early 70's they have evolved considerably, such that new solutions have appeared for computer networks mainly due to vague protocols specifications. We analyze a protocol extensively used today, the Dynamic Host Configuration Protocol (*DHCP*), in search of new forms of covert communication. From this analysis we observe several features that can be effectively exploited for subliminal data transmission. This results in the implementation of HIDE_DHCP, which integrates three covert channels that accommodate to different stealthiness and bandwidth requirements.
Keywords:Covert channels, System Information Security, Network Security

1 Introduction

Evolution of computer networks in recent years has led to the development of new services and, simultaneously, to the emergence of new threats for interconnected systems. *Covert Channels* is a sub-discipline of Information Hiding which has been usually considered a threat to security in both centralized (e.g. [14, 10]) and distributed systems (e.g. [15]). Network covert channels can be defined as a way of transmitting hidden information (i.e. unnoticed to a possible observer) using communication protocol features that are not properly defined or whose main functionality is misused. Therefore, as established, it should be possible to design covert channels at any level of the OSI stack, from the link layer to the application layer.

Traditionally, covert channels have been grouped into two main categories [1]: *storage* and *timing* channels. This classification refers to how the information to be transmitted is hidden. In the former, the sender hides data in memory areas to which the receiver has access. These memory areas might be certain header fields in network packets which are either obsolete or whose modification does not affect the proper functioning of the protocol. On the other hand, timing channels are based on modulating the behavior of the sender in order to encode information, which in practice is implemented packet rate alterations.

Covert channels have not only been studied from a theoretical perspective. Additionally, several tools have been designed to hide information flows between two or more hosts. Usually, these tools take advantage of protocols widely used in most existing networks, such as TCP/IP [19], HTTP [6] or DNS [13]. Nevertheless, there are still many protocols that have not been explored for covert communication channels. In this work we focus on the analysis and implementation of covert communications in the widely deployed DHCP protocol.

The rest of this paper is organized as follows. Section 2 provides an overview of the evolution of covert channels, from its origins to the present. Section 3 depicts a potential usage scenario, obtains its requirements and provides an exhaustive analysis on the opportunities for information hiding in DHCP. Next, Section 4 presents HIDE_DHCP, a new tool that integrates three forms of covert communications resulting from the previous analysis. Moreover, Section 5 presents the advantages and limitations of the implemented methods. Finally, Section 6 concludes this work and examines future research directions.

2 Related Work

The covert channel concept was first introduced by Lampson [14] in multilevel security systems to describe the ability of high security processes to signal information to other processes with lower security requirements. This concept began to draw security experts attention and was included in [1] and later in [10] for the evaluation of systems security. Also, a chapter of [18] was devoted to covert channels analysis and detection, where covert channels are first defined from a perspective that could be applied not only to multilevel security systems but also to computer networks.

Network covert channels were originally studied in [9], where two storage channels and one timing channel were identified. This work paved the way for new studies like [22], which analyses the IEEE 802.2 and 802.5 protocol families, and [11] which discusses the entire OSI reference model. The first known implementation, *Covert_TCP*, which is owed to Rowland [19] uses three methods to hide information in the TCP and IP headers[1]. Shortly after, *LOKI2* [4], a new covert channel that uses the payload in ICMP packets, was developed. Besides, *PingTunnel* [21] took advantage of ICMP to implement a subliminal channel. Some other well-known protocols were also exploited by *FirePass* [6] and *Ozyman* [13], which created covert channels in HTTP and DNS respectively.

In general, the aforementioned channels benefit from misused packet headers but some other authors developed timing channels as well. The first timing channel implementation is presented in [2], where the authors had to deal with synchronization problems due to the absence of a common precision clock. In [20], Shah et al. implement the *JitterBug*, which adds negligible delays to keystrokes in interactive network applications (e.g., telnet) and these delays conceal a message that is retrieved by a remote party. Also, [17] presents a highly reliable timing channel, *Cloak*, that encodes a message by a unique distribution of various packets over several TCP flows. In addition, many advances have been done on the detection of timing channels based on, for example, the similarity of time intervals [3] and the use of entropy and conditional entropy [7].

Although, extensive research has been conducted in the design and implementation of covert communication channels there are still numerous protocols which are susceptible to convey hidden data either for legitimate or illegitimate purposes. In particular, we evaluate the DHCP protocol which, to the best of our knowledge, has not been exploited for such purposes yet.

3 Covert Channel Analysis and Requirements

In this section we consider a fictitious setting from which we obtain the requirements for a new covert channel. After the identification of the needs, we perform an analysis on the potential provided by the protocol and devise several methods for information hiding.

[1] Even IPv6 was later found to be vulnerable to 22 forms of covert channels [16].

3.1 The Scenario

Consider a scenario where *Alice* and *Bob* want to communicate. Consider also that Alice works and has privileged access to part of a network deployed in the embassy of a country that is holding a summit of the Heads of State. Bob is visiting the embassy and has some information of extreme importance but they cannot be seen communicating or they would raise suspicion among the rest of participants. From this scenario we identify the following properties for the hidden communication channel:

- Stealthiness: This feature is leveraged by the fact that the protocol to be chosen has not been previously used to transmit covert data.
- Moderate bandwidth: The capacity is not a critical factor since the motivation of the channel is the transmission of small amounts of information, such as a cryptographic key.
- Reliability: The data being communicated is sensitive, therefore it is necessary that these are correctly received. The loss of small pieces of data may turn into a great loss of information.
- Locality. The goal is not to convey information through the Internet but to create a hidden communication channel between nodes in a local or personal area network, where there might be other entities monitoring the communications.
- Unidirectionality. The channel is not intended to allow the users to exchange information, thus having a one-way communication channel will suffice.

A suitable candidate given the above requirements is DHCP [5]. In the following we analyze the protocol in order to find the opportunity to conceal information within DHCP communications.

3.2 Protocol Analysis

DHCP (*Dynamic Host Configuration Protocol*), an auto-configuration protocol used on IP networks, can be considered as an extension of BOOTP (*Bootstrap Protocol*). It is an application-level protocol which uses UDP as its transport layer on ports 67 and 68 for the server and the client respectively. Regardless of using a datagram service, packet loss is unusual since the scope of this protocol is usually found within the same local area network. Despite this, DHCP provides some recovery mechanisms against packet loss, such as the retransmission of packets when no response is received after a given period of time.

The DHCP protocol uses a request-response model in which the client is always in charge of starting the communication. Client-server interaction is done in transactions, where several messages are exchanged. Table 1 provides a description of the different types of messages and the direction of the communication, where C and S represent the client and the server, respectively.

Two message exchange models exist in DHCP. The first model is used the first time a client requests the network configuration parameters, or in the case the configuration lease expires. Figure 1 depicts a complete configuration process. The second model comes into play if the lease is still valid but the client is trying to renew that specific configuration with the server. This model can be regarded as a sub-model of the previous one (see dashed arrows in Figure 1) since it starts with a Request message aiming to renew the configuration parameters with the DHCP server. The usual exchange is as follows: Discover, Offer, Request, Ack and, optionally (dotted arrow), Release. However, the first two messages are only possible in the first model.

Table 1: DHCP Messages

Packet	**Direction**	**Description**
Discover	C → S	Broadcast message used to find servers.
Offer	C ← S	Response to DHCP Discover. It contains a configuration offer.
Request	C → S	Message to confirm the acceptance of the parameters offered by the server or the renewal of a previous configuration.
Ack	C ← S	Message acknowledging the parameters agreed with the client. It includes the IP address to be used.
Nak	C ← S	Indicates the client the non acceptance of the configuration, either because the IP is incorrect or the lease has expired.
Decline	C → S	Message to indicate that the IP address is already in use by another client.
Release	C → S	Message to reject the given IP address, thus canceling the current lease.
Inform	C → S	Message used to request more information about the configuration; the client already has an IP address.

This is important because modifying the natural communication models might alert an observer of the presence of the channel.

DHCP messages share a common header structure regardless of whether they come from the server or the client. A detailed illustration of a DHCP packet is given in Figure 2. The packet is divided into various fields which are kept for backward compatibility with the original BOOTP protocol, thus providing more chances to allocate hidden data. In the following, we analyze the fields that we consider more relevant for covert communication purposes.

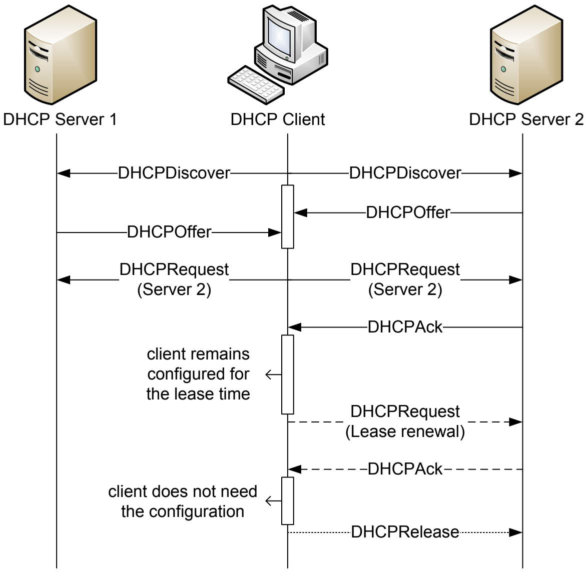


Figure 1: Potential exchange models between DHCP clients and servers

First, the transaction identifier (*xid*) field has the potential to convey up to 32 bits of covert information. Interestingly, according to the protocol RFC [5] this value must be randomly created by the client. This implies that there is no agreed algorithm for the identifier generation, as with the sequence number generator in TCP, which makes the detection of the covert channel more challenging.

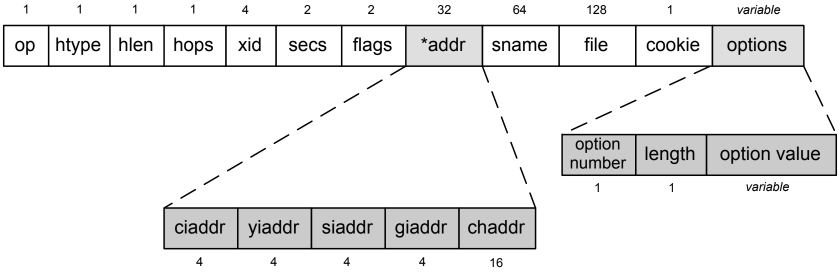


Figure 2: DHCP header format

The field *secs* can be used in a similar way as proposed in [8] to hide information in the TCP timestamps option. The low-order bit of TCP timestamps is basically random due to host internal timings, thus it is easy to change this value to convey data without alerting a potential observer. The concealment of information without a technique such as the one proposed in Griffin's work could raise suspicions or could cause a particular server to stop working correctly upon the reception of new messages apparently delayed in time.

Also, the field *chaddr* presents at least two potential ways of carrying hidden data. In the first place, it could be possible to use a technique similar to the one used by Rowland [19] in the third method developed for Covert_TCP, taking advantage of bouncing DHCP servers. In order to make this possible, the MAC address of the entity to be contacted should be included in *chaddr*. Besides, the data to be sent must be included in a header field which is not modified during the transaction, such as the *xid*. This makes the DHCP server to respond to the client specified in the *chaddr* instead of the real sender of the message, thus unknowingly helping the sender to convey the data to its destination. The only inconvenience would be having a datagram with a different MAC address to the one specified in the DHCP header. Although this is technically possible without disturbing the normal operation of the network, this could be easily identified as a spoofing attack by an intrusion detection system. The second chance for concealing data is due to the length of the *chaddr* field, which is 16 bytes long while most of the times it is used for Ethernet addresses (6 bytes). Since the amount of relevant data in *chaddr* is defined by the *hlen* field, the remaining bytes can be used to convey the covert data. These remaining bytes are to be considered by ordinary servers as garbage and they do not analyze them.

The fields *sname* and *file* are potentially excellent carriers of information due to their large size, 64 and 128 bytes respectively. Both fields consist of null-terminated strings ('\0'), thus our data might be included after this character without negatively impacting other clients or servers, which would consider that such fields do not contain relevant information. Although these fields are usually set to null by the operating system when they are not carrying their own data, in some situations they might contain information belonging to the Options field, which for reasons of packet capacity cannot be held within the field designed for such purpose. To indicate this situation, the RFC states that the option 52 (*Overload*) must be included. The main inconvenient in using these fields as covert information carriers is that, though not specified in the protocol definition, they are often filled with zero bytes when the *Overload* option is not active. This may alert to trained traffic analyzers.

Moreover, the *Options* field also presents interesting features which might be used to conceal information. The fact that it is a variable length field provides the ability to withhold much data, either on (1) the number of options used or (2) on the way options are ordered. Furthermore, we might also encode data by (3) placing a specific option in a particular position within the Options field. To clarify this, we provide further explanations on how each of the proposed methods could be used to encode the "ALOHA" string:

1. Number of options: for the sake of simplicity we assume that only capital letters can be sent, thus limiting the number of options to be included. To further reduce the number of options, instead of using the usual ASCII encoding we might use our own codification. For example, letter 'A' could be signaled by transmitting a message with 2 options[2], letter 'B' with 3 options, and so on. Therefore, to transmit "ALOHA" the client needs to send 5 DHCP messages, each of them with the following number of options: 2 ('A'), 13 ('L'), 16 ('O'), 9 ('H'), and 2 ('A') options respectively.

2. Options ordering: we assume that we want to encode the ASCII alphabet (8 bits) by setting options in a particular order. There are at least two ways of doing this. In the first case, we must have 8 reference options and if a particular option appears, the bit related to that option is equal to 1 and 0 otherwise. The second way is to have a reference message to compare with: if a particular option appears in the message received in the same order as in the reference message, the bit corresponding to this option is set to 1 and 0 otherwise. For example, assume we have an alphabet comprised of 16 symbols, then we should use 4 options to encode one character. Also assume that we considered the options A, B, C and D in that particular order. If we receive a message containing C, B, A, D, then the hidden message is 0101.

3. Option type: by using the type of an option placed in a particular position, we might encode an alphabet of up to 8 bits (options types are within the range 0 to 255). Without loss of generality, let us consider the option placed in the second position as the option used to conceal the covert data. In order to send the message "ALOHA" five messages are needed and, for each of them, the second option must be option number 65 (*"NIS-Server-Addr"*), 76 (*"STDA-Server"*), 79 (*"Service Scope"*), 72 (*"WWW-Server"*) and 65; which are the ASCII equivalent codes of the string to be sent. In order to increase the capacity of this channel, instead of encoding a single symbol per packet, several different options could be used as data carriers within every packet.

One of the biggest downsides presented by the use of the above methods is that depending on the type of message there are a number of options which are either mandatory or not permitted. Therefore, the ability to deploy any of these solutions will depend on whether the introduction of unauthorized options in specific packets will influence the operation of the protocol. Another drawback, which is specific to the third proposed method, is that encoding messages with repeated characters may entail the creation of repeated options within the same packet which, although possible in some cases, may raise suspicions. However, this issue might be easily solved by sending characters only if the next character to encode has not already been included in the current message.

Finally, one might take advantage of the existence of some particular options that are either undefined or declared for private use. These options (84, 96, 102-111, 115, 126, 127, 137-149, 151-174, 178-207, 212-219, 222 and 223 are not assigned or have been erased; and 224-254 are for private use), specially those for private use, are potentially excellent information carriers since their structure has not been defined. Thus, anyone could define the *value* field in the option to be as large as necessary, up to 255 bytes.

[2] The RFC requires the occurrence of at least 2 options: *"DHCP Message Type"* and *"End"*

4 HIDE_DHCP: A new subliminal channel

In this section we describe the implementation of HIDE_DHCP, which integrates 3 methods for covert communications using the *xid*, *Sname* and *File*, and *Options* fields. The implementation is based on an already existing DHCP code, that we modified, developed by the Internet Systems Consortium (ISC) numbered as 4.1.1-P1[12] and distributed in Linux operating systems.

4.1 Xid Implementation

The xid field is 4 bytes long and in the original ISC code it is loaded with the result of the *random()* function. This process is performed twice in the client code during the *make_discover* and *state_reboot* procedures, which are invoked every time a client requests a configuration.

The modifications performed on the ISC code were done in such a way that the protocol specification is not altered. This results in a stealthier channel which is fully compliant with any DHCP client or server. In particular, the modified clients are able to request for network configuration parameters while transmitting hidden information to the servers and also they might behave as usual clients without conveying any information at all. Consequently, this choice must be notified to the modified server in order to be able to interpret the *xid* as data. In order to help the server in identifying which client is sending covert data, *start* and *end* delimiters are used (codified within the *xid* field). These delimiters are predefined but they can be modified in order to introduce some uncertainty to potential network traffic analyzers.

Upon the reception of a packet coming from a colluding client and containing the *start* delimiter, a new covert session starts. The received data is stored locally until the reception of the *end* delimiter. The data contained in the *xid* field is retrieved from DHCP Requests since this is a common message in both exchange models (see Figure 1).

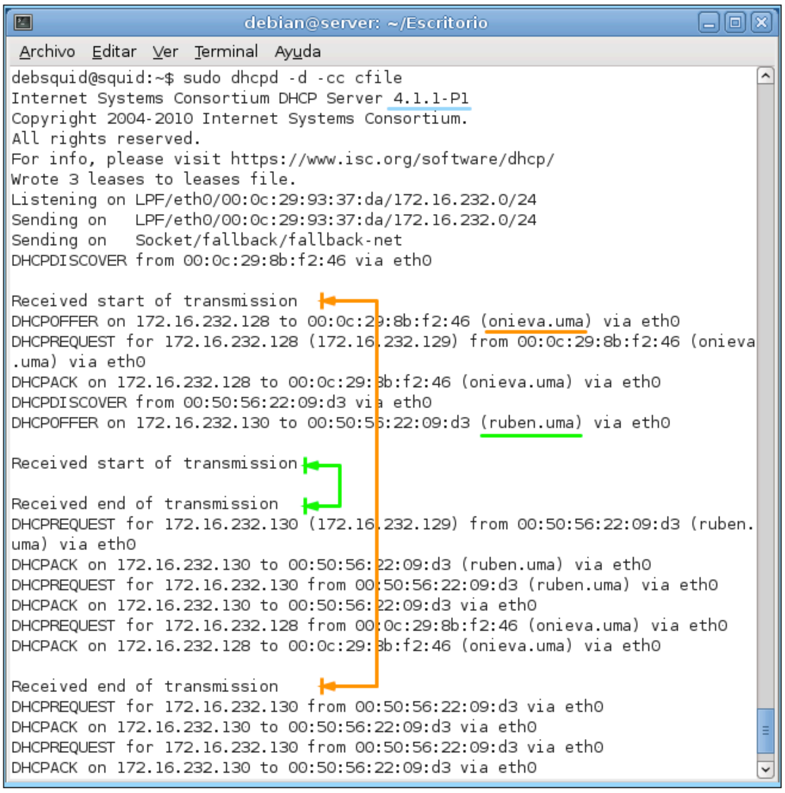
Furthermore, in order to enhance client-server interaction we introduce some mechanisms to allow these entities to determine if they are communicating with the right entity, since several clients and servers might coexist in the same network and sending covert data to unmodified servers is not only useless but also might increase the detection ratio. Therefore, if the client does not receive a reply from the expected server, then it will perform as an ordinary client.

4.2 Sname and File Implementation

Fields *sname* and *file* present very similar features. According to the protocol specification, these two fields are both defined as null-terminated strings and might only carry data in the case of Discover, Inform and Request packets coming from the client side, and Offer and Ack packets coming from the server side. This is taken into consideration in order to avoid altering the normal operation of the protocol.

The strategy in this case is to pretend to be sending empty fields by setting the first byte to the *null* character. Consequently, the implementation is able to hide a maximum of 190 bytes of data per packet, from which 63 bytes are kept within the *sname* field and 127 bytes within the *file* field. In this implementation Discover and Request packets are used as data carriers. This was not possible in the previous implementation because the *xid* field must be constant for a complete transaction. In Figure 3 we show a snapshot of a modified server simultaneously receiving data from two modified clients.

In this implementation we also make use of delimiters because some implementations do not set to *null* these fields but they insert garbage data instead. The use of delimiters prevent the modified server from storing undesired data.



```
debsquid@squid:~$ sudo dhcpd -d -cc cfile
Internet Systems Consortium DHCP Server 4.1.1-P1
Copyright 2004-2010 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Wrote 3 leases to leases file.
Listening on LPF/eth0/00:0c:29:93:37:da/172.16.232.0/24
Sending on   LPF/eth0/00:0c:29:93:37:da/172.16.232.0/24
Sending on   Socket/fallback/fallback-net
DHCPDISCOVER from 00:0c:29:8b:f2:46 via eth0

Received start of transmission
DHCPOFFER on 172.16.232.128 to 00:0c:29:8b:f2:46 (onieva.uma) via eth0
DHCPREQUEST for 172.16.232.128 (172.16.232.129) from 00:0c:29:8b:f2:46 (onieva
.uma) via eth0
DHCPACK on 172.16.232.128 to 00:0c:29:8b:f2:46 (onieva.uma) via eth0
DHCPDISCOVER from 00:50:56:22:09:d3 via eth0
DHCPOFFER on 172.16.232.130 to 00:50:56:22:09:d3 (ruben.uma) via eth0

Received start of transmission

Received end of transmission
DHCPREQUEST for 172.16.232.130 (172.16.232.129) from 00:50:56:22:09:d3 (ruben.
uma) via eth0
DHCPACK on 172.16.232.130 to 00:50:56:22:09:d3 (ruben.uma) via eth0
DHCPREQUEST for 172.16.232.130 from 00:50:56:22:09:d3 (ruben.uma) via eth0
DHCPACK on 172.16.232.130 to 00:50:56:22:09:d3 via eth0
DHCPREQUEST for 172.16.232.128 from 00:0c:29:8b:f2:46 (onieva.uma) via eth0
DHCPACK on 172.16.232.128 to 00:0c:29:8b:f2:46 (onieva.uma) via eth0

Received end of transmission
DHCPREQUEST for 172.16.232.130 from 00:50:56:22:09:d3 via eth0
DHCPACK on 172.16.232.130 to 00:50:56:22:09:d3 via eth0
DHCPREQUEST for 172.16.232.130 from 00:50:56:22:09:d3 via eth0
DHCPACK on 172.16.232.130 to 00:50:56:22:09:d3 via eth0
```

Figure 3: Simultaneous covert data reception

4.3 Options Implementation

From the various data hiding opportunities presented in Section 3.2 for the Options field we decided to use the options for private use. The main reason is that these might provide a substantial bandwidth.

The modified client injects its data in the payload of an option for private use[3]. In this case we selected option 224 and thus the use of delimiters is not required here because this option is not used otherwise. In case the server observes option 224 in the message coming from an colluding client, it stores the information.

Also, since the main advantage of this implementation is the bandwidth, we decided to include up to 255 bytes of covert data. In case the maximum capacity of a message is reached, the Overload option is used and the remaining options are included within the *sname* and *file* fields, if they are not being used.

5 Covert Channel Analysis

The covert implementations presented in Section 4 have different features that allows the user to decide which method to use depending on the circumstances. There is no best covert channel under all circumstances, and usually those who operate well in certain scenarios cease to be useful in others. Three properties stand out as the most important when dealing with covert channels: detectability, bandwidth and reliability. In our implementation, all the methods

[3] We noticed that some options for private use are being used in an unofficial way by many DHCP servers. A commonly used option is 252 (Proxy Autodiscovery), which is used for indicating the location of the proxy server configuration parameters

present full reliability, mainly due to the basic recovery mechanisms provided by DHCP against packet loss. However, the remaining properties differ between the three implementations.

The bandwidth is usually at odds with the ability to pass unnoticed and our solutions also present this trade-off. In terms of detectability, the three proposed methods have the advantage of being the first to use DHCP as a data carrier and moreover they strictly follow the protocol specification. Still, a smart observer might be alerted by some unusual patterns. In particular, the *xid* implementation is even stealthier because the original code fills the *xid* field with random data but the main downside of this method is that it provides a very limited bandwidth. Thus, this covert channel is recommended for small amounts of highly sensitive data, for example the transmission of an elliptic curve cryptography key.

The *sname* and *file* implementation provides a better bandwidth with a maximum capacity of 380 bytes in a full transaction (recall we hide data in Discover and Request packets). This result is significantly higher than the channel provided by the *xid* implementation (4 bytes). The main drawback is increased detectability. Even when both modified clients and servers use some techniques to prevent revealing the presence of the channel, there are some features that might arise suspicion on an experienced observer: in many implementations these fields are filled with zeros when not used. Besides, in order to allow modified clients to contact modified servers, their names are included in the *sname* field in every message coming from the server. By only making use of the *file* field we could create a bidirectional covert channel between clients a servers.

Finally, the *options* implementation provides the higher bandwidth and is recommended for loosely supervised networks because the size of the resulting DHCP messages might attract the attention of casual observers. This method would also allow the creation of a two-way covert channel because option 224 is not used for any other purposes. Also, the detectability of the channel might be improved by reducing the size of the option payload or by using different options to convey the data.

6 Conclusions

This paper presents an exhaustive analysis on the potential for covert communications in DHCP, a protocol that is extensively used for the configuration of IP-based devices. From this analysis we select and implement three storage channels that we integrate on a tool called HIDE_DHCP. These solutions present distinguishing features that makes them suitable under different circumstances. The main factors influencing the selection among the devised methods are the channel bandwidth and its detectability, which is made difficult by the fact that no previous implementations on DHCP exist and also because the normal operation of the protocol is not altered by the subliminal channel.

Several information hiding techniques have been discussed for DHCP but only a few of them have been implemented. We have further analyzed the advantages and limitations as well as possible means of improvement. Our current work goes in this direction. Also, performing more exhaustive detectability tests is a natural next step. Although we have performed some successful detectability tests with out-of-the-box intrusion detection solutions, much work can be done in this direction. Finally, our tests on a controlled LAN environment showed full reliability in terms of packet loss. However, running tests under extreme circumstances is also of interest.

Acknowledgments.

This work has been partially funded by the European Commission through the FP7 project NESSoS (FP7 256890) and the Spanish Ministry of Science and Innovation through the research projects: ARES (CSD2007-00004) and SACO (IPT-2011-1593-390000). The first author is supported by the Spanish Ministry of Education through the F.P.U. Program.

References

[1] Brand, S.L.: Department of Defense Trusted Computer System Evaluation Criteria, "The Orange Book". Tech. Rep. DoD 5200.28-STD, U.S. Department of Defense (1985), `http://csrc.nist.gov/publications/history/dod85.pdf`

[2] Cabuk, S., Brodley, C.E., Shields, C.: IP Covert Timing Channels: Design and Detection. In: Proceedings of the 11th ACM Conference on Computer and Communications Security. pp. 178–187. CCS'04, ACM Press, New York, NY, USA (2004)

[3] Cabuk, S., Brodley, C.E., Shields, C.: IP Covert Channel Detection. ACM Trans. Inf. Syst. Secur. 12, 22:1–22:29 (April 2009)

[4] Daemon9: Loki2 (the implementation) (1997), `http://www.phrack.org/archives/51/P51-06`

[5] Droms, R.: RFC 2131 - Dynamic Host Configuration Protocols (1997), `http://www.ietf.org/rfc/rfc2131.txt`

[6] Dyatlov, A.: Firepass - Gray-World.net Team (2003), `http://gray-world.net/it/pr_firepass.shtml`

[7] Gianvecchio, S., Wang, H.: An entropy-based approach to detecting covert timing channels. IEEE Trans. Dependable Secur. Comput. 8, 785–797 (Nov 2011)

[8] Giffin, J., Greenstadt, R., Litwack, P., Tibbetts, R.: Covert Messaging Through TCP Timestamps. In: Proceedings of the Privacy Enhancing Technologies Workshop (PET). pp. 194–208 (2002)

[9] Girling, C.G.: Covert Channels in LAN's. IEEE Trans. Software Eng. 13(2), 292–296 (1987)

[10] Gligor, V.D.: A Guide to Understanding Covert Channel Analysis of Trusted Systems, "The Light Pink Book". Tech. Rep. NCSC-TG-030, U.S. National Computer Security Center (1993)

[11] Handel, T.G., Sandford, M.T.: Hiding Data in the OSI Network Model. In: Proceedings of the First International Workshop on Information Hiding. pp. 23–38. Springer-Verlag, London, UK (1996)

[12] ISC: DHCP - Internet Systems Consortium, Inc. (2011), `http://www.isc.org/software/dhcp/`

[13] Kaminsky, D.: Tunneling Audio, Video, SSH and pretty much anything else over DNS (2004), `http://www.doxpara.com/`

[14] Lampson, B.W.: A Note on the Confinement Problem. Commun. ACM 16(10), 613–615 (1973)

[15] Li, S., Ephremides, A.: Covert channels in ad-hoc wireless networks. Ad Hoc Netw. 8, 135–147 (March 2010)

[16] Lucena, N.B., Lewandowski, G., Chapin, S.J.: Covert Channels in IPv6. In: Privacy Enhancing Technologies. LNCS, vol. 3856, pp. 147–166. Springer-Verlag, Berlin Heidelberg (2006)

[17] Luo, X., Chan, E., Chang, R.: Cloak: A ten-fold way for reliable covert communications. In: Biskup, J., Lopez, J. (eds.) Computer Security - ESORICS 2007, LNCS, vol. 4734, pp. 283–298. Springer Berlin / Heidelberg (2007)

[18] McHugh, J.: Covert Channels Analysis: A Chapter of the Handbook for the Computer Security Certification of Trusted Systems. Technical Memorandum 5540:062A, Naval Research Laboratory, Washington, D.C. (1996), `http://www.windowsecurity.com/uplarticle/12/COVCHAN.pdf`

[19] Rowland, C.H.: Covert Channels in the TCP/IP protocol suite (1996), `http://www.firstmonday.org/issues/issue2_5/rowland/`

[20] Shah, G., Molina, A., Blaze, M.: Keyboards and Covert Channels. In: USENIX-SS'06: Proceedings of the 15th Conference on USENIX Security Symposium. pp. 59–75. USENIX Association, Berkeley, CA, USA (2006)

[21] Stødle, D.: Ping Tunnel - Send TCP traffic over ICMP (2005), `http://www.cs.uit.no/~daniels/PingTunnel/`

[22] Wolf, M.: Covert Channels in LAN Protocols. In: LANSEC '89: Proceedings on the Workshop for European Institute for System Security on Local Area Network Security. pp. 91–101. Springer-Verlag, London, UK (1989)